On the Complexity of Join Predicates

Jin-Yi Cai * Venkatesan T. Chakaravarthy † Raghav Kaushik
Jeffrey F. Naughton

Dept. of Computer Sciences
University of Wisconsin, Madison WI 53706
Email: {jyc,venkat,raghav,naughton}@cs.wisc.edu

ABSTRACT

We consider the complexity of join problems, focusing on equijoins, spatial-overlap joins, and set-containment joins. We use a graph pebbling model to characterize these joins combinatorially, by the length of their optimal pebbling strategies and computationally, by the complexity of discovering these strategies. Our results show that equijoins are the easiest of all joins, with optimal pebbling strategies that meet the lower bound over all join problems and that can be found in linear time. By contrast, spatial-overlap and set-containment joins are the hardest joins, with instances where optimal pebbling strategies reach the upper bound over all join problems and with the problem of discovering optimal pebbling strategies being NP-complete. For set-containment joins, we show that discovering the optimal pebbling is also MAX-SNP-Complete. As a consequence, we show that unless NP = P, there is a constant ϵ_0, such that this problem cannot be approximated within a factor of $1 + \epsilon_0$ in polynomial time. Our results shed some light on the difficulty the applied community has had in finding "good" algorithms for spatial-overlap and set-containment joins.

1. INTRODUCTION

The "join" operation has received a great deal of attention both from database researchers and from database system implementers. Recall that in relational algebra, the join operation can be expressed as a join selection predicate on the cross product of two relations. The most studied and common join is the equijoin, in which the join predicate is equality. However, more recently, with the advent of DBMSs with extensible type systems, research has turned to joins with other classes of join predicates, including spatial overlap joins and set containment joins. In this paper, we ask if there is anything intrinsically harder about joins with different classes of join predicates.

Our motivation for this work comes from a growing intuition that spatial overlap and set containment joins really are harder than equijoins. For equijoins, there are a number of recognized good algorithms, including index nested loops, sort-merge join, and hash-join; since these algorithms were first proposed, subsequent research has focussed on optimizations of these algorithms rather than on new algorithms. However, when we turn to spatial overlap joins and set containment joins, the situation is different. While there are a number of algorithms for these joins, at some level, they are not as satisfying as the equijoin algorithms, requiring either replication of data or repeated processing of data. Spatial join and set-containment joins cannot be considered to be solved problems.

Of course, from a superficial perspective, these joins are clearly harder — they are more complicated. Our goal here is to make this statement precise in a formally provable way. It turns out that the simple problem of joins with different classes of join predicates has a surprisingly rich structure. While our goal is not to devise new algorithms for these problems, our results do shed some light on the problems the community has had in devising good algorithms for spatial overlap and set-containment joins.

To compare the difficulty of different join predicates, we model the fundamental operations in a join computation as a pebbling game, and analyze this pebbling game. Our results show that equijoins are the easiest of all joins both combinatorially and computationally, with optimal pebbling strategies that meet the lower bound over all join problems and that can be found in linear time. By contrast, we show that spatial-overlap and set-containment joins are the hardest joins, with instances where optimal pebbling strategies reach the upper bound over all join problems, and with the problem of discovering optimal pebbling strategies being NP-complete. For set-containment joins, we show that discovering the optimal pebbling is also MAX-SNP-Complete. As a consequence, we show that unless NP = P, there is a constant ϵ_0, such that this problem cannot be approximated within a factor of $1 + \epsilon_0$ in polynomial time.

*Supported in part by NSF grant CCR9820806 and by a Guggenheim Fellowship
†Supported by NSF grant CCR9820806

PODS '01 Santa Barbara, California USA

2. THE PEBBLE GAME MODEL

In this paper, for simplicity, we assume that all relations have a single column, and that all joins are on that column. The relations are allowed to be multi-sets. We consider join problems defined as follows: Given two relations $\mathcal{R}(A)$ and $\mathcal{S}(B)$ and a join predicate $\bowtie$, generate pairs of tuples (r, s), $r \in \mathcal{R}$ and $s \in \mathcal{S}$, such that $r \bowtie s$ holds. Perhaps the simplest join is the equijoin, in which $r \bowtie s$ if $r.A = s.B$. Equijoins in relational systems are an extremely well-studied problem; see [2] for a good overview.

For equijoins, A and B can be over any domain that supports equality. In traditional relational systems, these domains are either character strings or are some flavor of numeric type. Recently, with the advent of object-relational DBMS [15], researchers have begun considering (and developers have begun implementing!) domains of other types. These new types include spatial types, in which the elements of the domain are typically polygons over some coordinate system; and set-valued types, in which the elements of the domain are sets.

Over these new domains, researchers have begun investigating algorithms to support new join predicates. In spatial domains, the most common join considered has been polygon overlap [3, 8, 13] (here, $r.A \bowtie s.B$ if the polygon in $r.A$ overlaps the polygon in $s.B$.) In set-valued domains, the most common predicate considered has been set-containment [5, 14], in which $r.A \bowtie s.B$ if $r.A \subseteq s.B$.

Since our goal is to study the join problems themselves, and not specific algorithms for their evaluation, we need an abstract model for join computation that is independent of any algorithm. In our work we model an instance of the join problem as a bipartite graph $G = (R, S, E)$, where R has one vertex for every tuple in $\mathcal{R}$ and S has one vertex for every tuple in $\mathcal{S}$. Vertices $u \in R$ and $v \in S$ are connected by an edge in E if the corresponding tuples join under the join predicate. We call this the *join graph* of this instance.

For every pair of tuples (r, s) that joins, *any* join algorithm has to consider this pair of tuples at some point of time in its execution and produce a result tuple. We model this by stating that the join algorithm places one pebble on each vertex that corresponds to r and s in the join graph and removes the edge between them. Performing a join can then be modeled as a sequence of moves of pebbles in the join graph, the purpose of which is to delete all edges.

Clearly, this simple abstract model does not model all of the costs in a join algorithm (although the "merge" phase of a sort-merge join does in some sense resemble this pebbling game). This is intentional; recall that our goal is to explore differences in the intrinsic difficulty of join problems rather than to explore the performance of specific algorithms.

The pebbling game can be described as follows. We are given a bipartite graph[1] $G = (R, S, E)$, where R and S are the two partitions and $E \subseteq R \times S$. We are also given two pebbles, which can be placed on nodes in the graph. We start with no pebbles on the graph. When the two pebbles are on the incident vertices of an edge, the edge is deleted. In a single move, one of the two pebbles can be moved to another node. A *pebbling scheme* is a sequence of moves that deletes all edges. We denote a pebbling scheme by a sequence of *pebbling configurations* $P = p_1, p_2, \ldots p_k$, where each p_i is a pair of nodes (u, v).

Our pebbling game deals with only the edge set of G, hence we will remove *a priori* all isolated vertices, and assume henceforth that all G in this paper have no singletons. The input size for our problem is m, the number of edges in G. This is also the number of tuples produced by the join. Thus, our results are expressed in terms of the output size.

In related work, a similar pebbling game was considered in [6]. There, the nodes of the graph were disk pages of tuples, and the pebbling cost was used to capture the I/O cost of scheduling page fetches for this specific layout of disk pages. The main result of that paper was that the problem of finding the optimal pebbling scheme is NP-Complete. It was shown in [7] that finding the optimal pebbling scheme for spatial joins is NP-Complete. This was also in the context of scheduling page-fetches for a specific layout of disk pages. These results imply Theorem 4.2. However, their work differs from ours in that, in their approach there was no notion of the inherent complexity of a join predicate. Rather, the focus was on the problem of scheduling page-fetches. In particular, they did not consider upper and lower bounds on the length of optimal pebbling sequences, nor did they use their model to investigate the complexity of different classes of join predicates.

2.1 Cost Models

In this section, we describe our cost model and some of its useful properties.

DEFINITION 2.1.: *Let G be a bipartite graph and $P = p_1, p_2, \ldots p_k$ be a pebbling scheme for G. The cost of P is $\hat{\pi}(P) = k + 1$, the number of pebble moves (1 is added to account for the initial placement of a pebble). The optimal pebbling cost for G is denoted by $\hat{\pi}(G)$.*

In this formulation of the problem by a pebbling game, the essential input to the problem is the edge set of the graph, while the vertex set is secondary, and isolated vertices are removed. If G is disconnected, then any pebbling scheme has to pay a cost of one for placing a pebble in a new component, which more or less represents a start up cost. To reflect more intrinsically the cost of a good pebbling scheme, we define

DEFINITION 2.2.: *The effective cost of a pebbling scheme P, $\pi(P)$, is $\pi(P) = \hat{\pi}(P) - \beta_0(G)$, and $\pi(G) = \hat{\pi}(G) - \beta_0(G)$, where $\beta_0(G)$ is the number of connected components in G, a.k.a. the 0th Betti number.*

In every move, at most one edge can be deleted, and in an optimal scheme, at most two moves are required to delete a given edge. Hence,

LEMMA 2.1.: *If G is a graph with m edges, then $m + 1 \leq \hat{\pi}(G) \leq 2m$*

[1] This definition applies for general graphs as well.

COROLLARY 2.1.: *For a connected graph G with m edges, $m \leq \pi(G) \leq 2m - 1$.*

A disconnected graph with several connected components represents essentially separate join problems. The following *additivity lemma* states that there is nothing to be gained in our model by lumping these problems together.

LEMMA 2.2.: *The disjoint union $G \cup H$ of two bipartite graphs G and H satisfies:*

$$\hat{\pi}(G \cup H) = \hat{\pi}(G) + \hat{\pi}(H)$$
$$\pi(G \cup H) = \pi(G) + \pi(H)$$

PROOF: First of all, it is easy to see that $\hat{\pi}(G \cup H) \leq \hat{\pi}(G) + \hat{\pi}(H)$. We now have to show that $\hat{\pi}(G \cup H) \geq \hat{\pi}(G) + \hat{\pi}(H)$. Let P be an optimal pebbling scheme for $G \cup H$. Without loss of generality, let us assume that the first edge deleted by P is an edge in G. We claim that there is a pebbling scheme P' for $G \cup H$ which deletes all edges of G before deleting any edge of H, and $\hat{\pi}(P') \leq \hat{\pi}(P)$. The key observation is that for any pair of pebbling configurations $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, where x_1, y_1, x_2, y_2 are all distinct, it takes at least two moves to change the configuration from p_1 to p_2. Let e_1 be the first H-edge deleted in P. If there are G-edges deleted in P after e_1, let the first such be e_2. Let e_3 be the last G-edge deleted after e_2 before any more H-edges are deleted. The run of edges between e_2 and e_3 are all G-edges. Now modify P so that this run $e_2 \ldots e_3$ is placed just before e_1. Since G and H are disconnected, moving from an edge in one component to another takes at least two moves, so the costs associated to e_1 and e_2 in P (i.e. the cost of changing the pebbling configuration to e_i from the previous configuration) were at least 2. Hence the new scheme has cost no more than P, yet has more G-edges removed before any H-edge is removed. Proceeding thus, we can see that there is an optimal pebbling scheme P' which deletes all G-edges before moving on to H-edges. □

Given the additivity lemma (Lemma 2.2), in the rest of the paper, we focus on connected graphs.

From the additivity lemma and Corollary 2.1 it follows that

LEMMA 2.3.: *For a graph G with m edges, $m \leq \pi(G) \leq 2m - 1$*

DEFINITION 2.3.: *We say that G has a perfect pebbling scheme, if $\pi(G) = m$, where m is the number of edges in G.*

LEMMA 2.4.: *If G is a matching with m edges, then $\hat{\pi}(G) = 2m$ and $\pi(G) = m$.*

2.2 Relationship of Pebble Game to TSP

A pebbling scheme "moves" from one edge to the next. If we view the edges as "abstract" nodes, it is in some sense a traveling salesman path. To capture this intuition, we use the notion of line graphs [4]. The line graph $L(G)$ of a graph G is a graph in which each edge in G is represented by a node. Two nodes in $L(G)$ are adjacent iff the corresponding edges in G share an end point.

PROPOSITION 2.1. *Let G be a connected (bipartite) graph with m edges. Then $\pi(G) = m$ iff $L(G)$ has a Hamiltonian path.*

PROOF: Recall that G has a perfect pebbling scheme if $\pi(G) = m$. Let P be a perfect pebbling scheme for G. Then every move deletes an edge. This sequence of edges constitutes a Hamiltonian path in $L(G)$. Conversely, let $L(G)$ have a Hamiltonian path. View the sequence of edges in the path as a sequence of pebbling configurations. We get a perfect pebbling scheme for G. □

In order to generalize the above proposition for imperfect pebbling schemes, we view $L(G)$ as a weighted complete graph. The weight between two nodes is set to one if there is an edge between them and two, otherwise. In this "completed" $L(G)$, any traveling salesman tour never needs to visit a node more than once. Thus, in the rest of the paper, we use the term *TSP tour* to mean a sequence of visits to every node exactly once.

PROPOSITION 2.2. *The optimal TSP tour in $L(G)$ has cost exactly $\pi(G) - 1$*[2].

We omit the proof here.

In $L(G)$, let us call the edges of weight one *good* and edges of weight two *bad*. If a TSP tour moves along a bad edge, it is said to *jump*. The cost of any TSP tour is $m - 1 + J$, where m is the number of nodes in $L(G)$ and J is the number of jumps. J is said to be the *extra cost* of the tour.

3. COMBINATORIAL BOUNDS ON THE PEBBLING COST

We examine the structure of the join graphs of the following predicates: 1) equijoins, 2) spatial overlap joins, and 3) set containment joins and establish combinatorial bounds on the optimal pebbling cost.

Before moving on to the details of the join graphs for the different predicates, let us observe the following result for an arbitrary bipartite graph (in fact, the following proof is valid for a general connected graph that need not be bipartite).

THEOREM 3.1.: *Let G be a connected bipartite graph. Then $\pi(G) \leq \lceil 1.25m \rceil - 1$, where m is the number of edges in G.*

PROOF: We show that $L(G)$ has a TSP tour of cost at most $\lceil 1.25m \rceil - 2$. We give a partition $E = E_1 \cup \ldots \cup E_k$, where each E_i has a Hamiltonian path and at most one $|E_i| < 4$ (E is the set of nodes in $L(G)$). Without loss of generality, $|E| \geq 4$ since all connected graphs of order at most 3 have a Hamiltonian path. It is known that $L(G)$ is connected given G is connected, and $K_{1,3}$ is not an induced subgraph of $L(G)$ [4].

[2]The difference by 1 is merely due to the way the length of a TSP tour is typically measured, namely the first vertex of the tour counts 0.

Hence, in any (rooted) Depth First Search (DFS) tree T of $L(G)$, any node has at most two children. We call a pair of nodes *twins* if they are both leaves and share the same parent. If l_1, l_2 are twins, let p be their parent and g be the grandparent. By being $K_{1,3}$-free and since $|E| \geq 4$, g exists, and either l_1 or l_2 is adjacent to g. Without loss of generality it is l_1. Remove the edge (g, p) and add (g, l_1) in the DFS tree T. Note that, by repeating this procedure we obtain a tree T' which has no twins and where every node has at most two children. Consider all the nodes that have at least four descendants (including itself). Among all such nodes pick one at the lowest level and call it r. It can be verified that the subtree rooted at r is a path and hence is Hamiltonian. Remove the nodes in this subtree from $L(G)$. The resulting graph is still connected. By repeating this procedure, we can obtain the desired partitioning of the nodes in $L(G)$.□.

While the above construction yields an algorithm that is linear in the size of the line graph, an alternative proof of the above theorem provides us with a linear time algorithm to achieve this bound.

LEMMA 3.1.: *Given a connected bipartite graph G with m edges, finding a pebbling scheme with cost $\leq \lceil 1.25m \rceil$ can be done in linear time.*

3.1 Equijoins

Let us first consider equijoins. Every connected component in the join graph for equijoins is a complete bipartite graph.

LEMMA 3.2.: *If G is a complete bipartite graph with m edges, then $\pi(G) = m$.*

PROOF: Let G be a $k \times l$ complete bipartite graph. Let the vertices on one side be $u_1, \ldots, u_k$ and on the other side be $v_1, \ldots, v_l$. Then the sequence of pebbling configurations: (u_1, v_1),(u_1, v_2),...,(u_1, v_l),(u_2, v_l),(u_2, v_{l-1}), ...,(u_2, v_1), (u_3, v_1),... pebbles G with cost m, where m is the number of edges in G. □

Now, let us examine the optimal pebbling cost of the join graph for an equijoin.

THEOREM 3.2.: *Let G be the join graph of an equijoin, with m edges. Then, $\pi(G) = m$. That is, the join graph for equijoins can be pebbled perfectly.*

PROOF: No pebble moves are needed for isolated nodes. By Lemma 3.2 every connected component in the join graph can be pebbled perfectly. The theorem follows from Lemma 2.2 □

3.2 Set Containment Joins

Next, we turn to set containment joins. These joins are universal in the following sense.

LEMMA 3.3.: *Given any bipartite graph $G = (R, S, E)$, there is an instance of the set containment join problem such that G is its join graph.*

PROOF: Let $R = \{r_1, r_2, \ldots, r_k\}$ and $S = \{s_1, s_2, \ldots, s_l\}$. Consider the instance of set containment joins where r_i denotes the tuple $\{i\}$, and s_j denotes the tuple

$$\{i : (r_i, s_j) \in E\}$$

We can see that the join graph for this instance of set containment joins is G. □

Thus, the lower and upper bounds on the pebbling cost of set containment joins is the same as the bounds for general bipartite graphs. As shown in Theorem 3.1, any connected bipartite graph can be pebbled in 1.25 times the number of edges.

On the other hand, there are bipartite graphs that require the above bound as we now show.

THEOREM 3.3.: *There is a family of bipartite graphs $\mathcal{F}$, such that for any graph $G \in \mathcal{F}$,*

$$\pi(G) = \lceil 1.25m \rceil - 1$$

PROOF: Consider the family of bipartite graphs $\mathcal{F}$=$\{G_3, G_4, G_5, \ldots\}$, where G_3, G_4, G_5 are shown in Fig 1(a), and G_n for a general n is built similarly. Let G be any graph in this family.

By Theorem 3.1, we know that $\pi(G) \leq \lceil 1.25m \rceil - 1$, where for G_n, $m = 2n$. We now show that $\pi(G) \geq \lceil 1.25m \rceil - 1$. We do this by proving that in the line graph of G, $L(G)$, viewed as a complete weighted graph as described above, any TSP tour must cost at least $\lceil 1.25m \rceil - 2$.

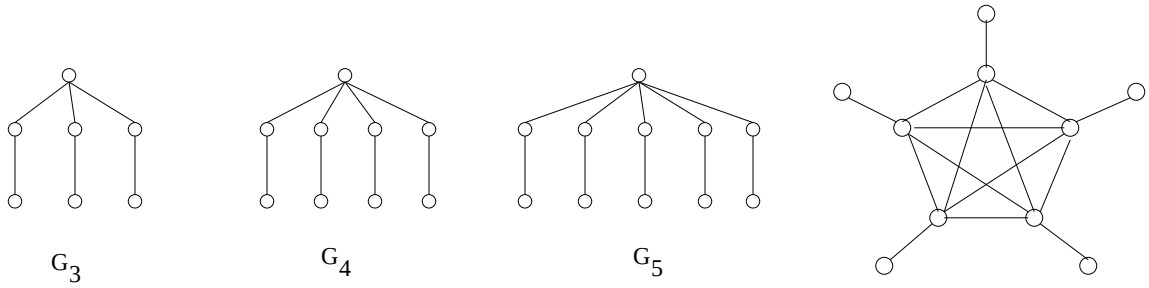
The line graph for G_5 is shown in Figure 1(b). In general, $L(G_n)$ is K_n with n extra nodes of degree 1, each connected to the n nodes of K_n in a 1-1 fashion. Consider a TSP tour of $L(G_n)$. Let $B^+ = \{$nodes entered via a bad edge$\}$, and $B^- = \{$nodes left via a bad edge$\}$. The TSP tour of $L(G_n)$ has cost $m - 1 + J$, where $J = |B^+|$ as well as $J = |B^-|$. Hence, $2J = |B^+| + |B^-| \geq |B^+ \cup B^-|$. Now $L(G)$ has $n = m/2$ vertices of degree one. Any TSP tour should either enter or exit each of these leaves by a bad edge, except for the very first and last node of the tour. It follows that $|B^+ \cup B^-| \geq m/2 - 2$. Thus, $2J \geq m/2 - 2 \Rightarrow J \geq \lceil m/4 \rceil - 1$. So the total cost of the tour should be at least $m - 1 + J \geq \lceil 1.25m \rceil - 2$. □

We already noted that this worst case bipartite graph can be realized as a set containment join. Note that the above graph cannot be the join graph for an equijoin since it is not a complete bipartite graph. Thus, unlike equijoins where there is always a perfect pebbling scheme, there are instances of set-containment joins where any optimal algorithm to produce a pebbling scheme, irrespective of its computational complexity, yields a pebbling cost of 1.25 times the number of edges.

3.3 Spatial Overlap Joins

Next we turn to spatial overlap joins. By the above upper bound result on general join graphs, we know that any spatial join graph can be pebbled in 1.25 times the number of edges. It turns out that there is an instance of the spatial join whose join graph has worst case behavior. Hence, even spatial overlap joins are also inherently more difficult than equijoins by a purely combinatorial metric.

LEMMA 3.4.: *There is a family of instances of the spatial overlap join problem whose join graphs are the ones shown in Fig 1(a).*



(a)
Family of Graphs

(b)
Line Graph for G_5

Figure 1: Example for Lower Bound

4. FINDING THE OPTIMAL PEBBLING SCHEME

We now proceed to classify the join predicates under consideration by the *computational complexity* of *finding* the optimal pebbling scheme.

DEFINITION 4.1.: PEBBLE *is the following problem: Given a bipartite graph G, find the optimal pebbling scheme. For a given $\epsilon > 0$, the ϵ-approximation problem is to find an approximation to $\pi(G)$ within a factor $1 + \epsilon$. The decision version PEBBLE(D) is: Given G and integer K, decide whether $\pi(G) \leq K$.*

THEOREM 4.1.: PEBBLE *can be solved in linear time for equijoin graphs.*

We observe that the construction given in Theorem 3.2 is similar to the merge phase of sort-merge join. For a general join graph, *PEBBLE* turns out to be hard. It does not become easier even if the input graph is known to be the join graph of spatial overlap join. This follows from [6] and [7].

THEOREM 4.2.: *PEBBLE(D) is NP-Complete [6]. PEBBLE(D) remains NP-Complete even if the input graph is known to be a spatial overlap join graph [7].*

The first part of Theorem 4.2 follows directly from [6], although their motivation was different from ours, in that they were investigating the scheduling of page reads during joins rather than considering computing joins over individual tuples with different classes of join predicates. In [7], the authors considered scheduling page reads, but in their case they assumed that the pages were rectangles partitioning a coordinate system. As this is a special case of the spatial overlap join we consider, their result implies the second part of preceding theorem.

The question now arises whether we can find approximation algorithms for *PEBBLE* within any constant factor $1 + \epsilon$ in polynomial time. By Lemma 3.1, we have a linear time approximation algorithm that finds a pebbling scheme that is within a factor of 1.25 from the optimal. With more work, one can approximate better. In particular, we note that an algorithm by Papadimitriou and Yannakakis can be used to approximate *PEBBLE* within a factor of 7/6 [12].

An approximation problem is said to have a *Polynomial Time Approximation Scheme* (PTAS) if there is a polynomial time algorithm such that for every constant $\epsilon > 0$, there is a polynomial p_ϵ, such that for any problem instance x, it finds an approximate solution to the problem within $1 + \epsilon$ (for minimization problems) of the optimal solution, in time $p_\epsilon(|x|)$. Note that NP-hardness does not imply either the existence or the non-existence of PTAS, even assuming NP $\neq$ P.

The complexity class MAX-SNP was defined by Papadimitriou and Yannakakis [11] and is somewhat technically involved (see [9]). A problem in MAX-SNP is MAX-SNP-Complete, if every problem in MAX-SNP is reducible to it by a certain L-reduction. If a problem A is L-reducible to a problem B, then any PTAS for B gives a PTAS for A. In particular, if one knows that A does not possess a PTAS, then neither does B. The PCP theory [1] proves that no MAX-SNP-Complete problem has a PTAS, assuming NP $\neq$ P. We show that *PEBBLE* is MAX-SNP-Complete.

Membership in MAX-SNP for *PEBBLE* can be shown directly, or by noticing that *PEBBLE* already has a constant factor approximation algorithm, which, for problems in NP, is known to imply membership in MAX-SNP [9].

We next define L-reduction, which is basically a reduction that preserves approximability.

DEFINITION 4.2.: ([9]) *Let A and B be two optimization problems. An L-reduction from A to B is a pair of functions f and g, both computable in polynomial time, such that there exist constants $\alpha, \beta > 0$,*

1. *If x is an instance of problem A, then $f(x)$ is an instance of B, such that*

$$OPT(f(x)) \leq \alpha OPT(x),$$

where OPT denotes the optimal cost;

2. *If s is any feasible solution of $f(x)$, then $g(s)$ is a feasible solution of x such that*

$$|OPT(x) - Cost(g(s))| \leq \beta |OPT(f(x)) - Cost(s)|.$$

Our proof of MAX-SNP-Completeness for *PEBBLE* is an L-reduction from a known MAX-SNP-Complete problem. Let TSP-(1,2) be the problem of finding the optimal traveling salesman tour in a complete graph, where the edge weights are either 1 or 2. Let TSP-k(1,2) be the same problem, where each node in the graph has at most k incident edges of weight 1. Let HAM-PATH-k be the Hamiltonian path problem on graphs with bounded degree of k. It is known that TSP-4(1,2) is MAX-SNP-Complete [12]. We first give an L-reduction from TSP-4(1,2) to TSP-3(1,2). Then we give an L-reduction from TSP-3(1,2) to *PEBBLE* via line graphs.

THEOREM 4.3.: *TSP-3(1,2) is MAX-SNP-Complete.*

PROOF: TSP-4(1,2) was shown to be MAX-SNP- Complete in [12]. We L-reduce this problem to TSP-3(1,2). HAM-PATH-3 is shown to be NP-Complete by giving a reduction from HAM-PATH-4 in [10] using the gadget shown in Fig 2. We use this gadget in our L-reduction. We call this gadget a *diamond*. We call the nodes a,b,c and d *corner* nodes and the others, *central* nodes. Notice that in the diamond, a Hamiltonian path exists between any two corner nodes and any Hamiltonian path in the diamond should start and end in corner nodes.

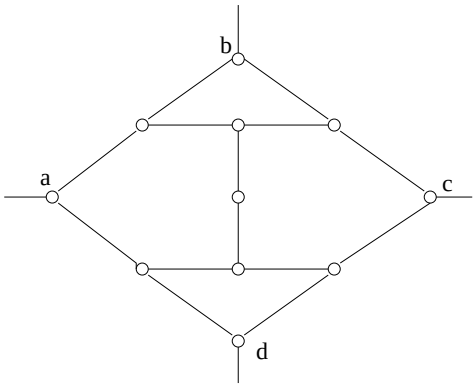


Figure 2: Gadget used to show TSP-3(1,2) is MAX-SNP-Complete

Given an input graph G, degree bounded by four, we obtain a graph $f(G) = H$, by replacing every node u of degree four by a diamond and connecting each of the four edges to exactly one corner. We call this diamond d_u.

We first show that $OPT(H) \leq 11OPT(G)$. Let T be the optimal TSP tour in G. We prove the above bound by exhibiting a tour T' of H within the above bound. We replace each node u of degree 4 in T with a suitable Hamiltonian path from corner c_1 to corner c_2 of the diamond corresponding to u. The two (distinct) corners c_1 and c_2 are chosen as follows. In T, let the node before u be x and the one after it be y. If (x, u) is a good edge let c_1 be the corner corresponding to x. Otherwise choose c_1 to be any of the corners. c_2 is chosen similarly.

It is easy to see that the extra cost (number of jumps) of T' is at most that of T. Since the number of nodes in H is at most $11n$, where n is the number of nodes in G, $OPT(H) \leq 11OPT(G)$.

We next exhibit the reduction g, which given a TSP tour T of H, produces a TSP tour T' of G, preserving approximability. We call T *nice* with respect to a node u of G, if the tour visits all the nodes of d_u (the diamond corresponding to u) consecutively. The tour is called *nice* if it is nice with respect to every node of G. We show that T can be converted into a nice tour without increasing its cost.

Let u be an arbitrary node of degree 4 in G. We convert T into a tour T_u that is nice with respect to u by the following procedure. Denote the segment of T between two nodes p and q, by $p-q$ segment. A d_u-segment of T is one where all the nodes are from d_u. A segment s of T is called *perfect* if it every edge in s is good and s is entered and left through good edges.

The procedure considers all the d_u-segments in T and chooses a segment s as follows. If a perfect d_u-segment is available, it is chosen as s (if more than one is available any one is chosen). Otherwise, s is chosen arbitrarily. Let the entry and exit points for s be a and b. Let the node appearing before a in T be p and the one appearing after b be q. Two distinct corner points c_1 and c_2 are chosen and s is replaced by the Hamiltonian path from c_1 and c_2. The procedure sets $c_1 = a$ if a is a corner point and $c_2 = b$ if b is a corner point. c_i are chosen arbitrarily otherwise. All the other d_u-segments in T are removed. This modified tour T_u is certainly nice with respect to u.

The chosen segment s may or may not be perfect. In either case, the extra cost in the $p - q$ segment of T_u is no more than its extra cost in T. Now let us consider the extra cost incurred by bypassing deleted d_u-segments. If s is not perfect, then all the other d_u-segments are also not perfect. Bypassing these does not increase the extra cost in T_u. Consider the case where s is perfect. Then there can be at most one other perfect d_u-segment s'. Assume such an s' exists. Bypassing this may add an additional jump in T_u. But, by examining the gadget, we find that no two perfect segments can cover all the nodes in the gadget. Thus, there has to be at least one d_u-segment s'' that is not perfect. Being perfect, s and s' use all the four corner points and hence, s'' has to be entered and exited via jumps. Bypassing s'' saves one of these jumps and that compensates for the (potentially) additional jump incurred in bypassing s'. If there are any other d_u-segments (other than s, s' and s''), they are all imperfect and removing them does not add any additional jump in T_u.. The only case left to be considered is when s is perfect and there is no other perfect d_u-segment. Bypassing the imperfect segments does not add any extra jumps.

We have a nice tour with respect to one node u. By repeating this process for all nodes we get a nice tour T'' of H. Now we obtain a tour $T' = g(T)$ of G by visiting the nodes in the same order in which the diamonds appear in T''. It is

easy to see that the extra cost of T' is no more than that of T''. Hence we have an L-reduction with $\alpha = 11$ and $\beta = 1$. □

This result has independent interest in Theory. It is known that HAM-PATH-3 is NP-Complete. This theorem shows that the corresponding TSP version of the problem is MAX-SNP-Complete, improving the result that TSP-4(1,2) is MAX-SNP-Complete.

THEOREM 4.4.: PEBBLE *is MAX-SNP-Complete.*

PROOF: A problem in NP is in MAX-SNP iff it has some constant factor approximation algorithm [9]. By Theorem 3.1 we have an approximation algorithm that does this. We now L-reduce TSP-3(1,2) to *PEBBLE*.

This reduction uses concepts from [6]. Given a TSP-3(1,2) graph $G = (V, E)$, $f(G)$ outputs the incidence graph of G, namely, the bipartite graph $B = (X, Y, E')$, where $X = V$ and $Y = E$. A node e in Y is joined with a node x in X, if the edge e in G has x as one of its end points.

We claim that f satisfies the first property of L-reductions. Let the line graph of B be $L(B)$. Observe that it can be obtained directly from G, by replacing every vertex of degree $i \in \{1, 2, 3\}$ by a clique of i vertices, where each of the i edges is connected to exactly one of the i vertices of the clique. First note that, if the optimal tour in G is of length c, then we can obtain a tour of length at most $3c$ in $L(B)$. This tour of $L(B)$ can be translated into a equally good pebbling strategy for B via Proposition 2.2. Therefore, f satisfies property 1 in Def 4.2 with $\alpha = 3$.

Now, given a pebbling strategy for B, it can be first translated into an equally good TSP tour T for $L(B)$. This tour can be converted into a tour T' of G, with $\beta = 1$, by a process similar to the one in Theorem 4.3. □

COROLLARY 4.1.: *TSP-3(1,2) for line graphs is MAX-SNP Complete.*

The consequence of MAX-SNP-Completeness for *PEBBLE*, following PCP theory, is that assuming NP $\neq$ P, there is some absolute constant $\epsilon_0 > 0$, such that there is no polynomial time approximation algorithm for *PEBBLE* within $1 + \epsilon_0$. This is a stronger statement than the non-existence of a PTAS.

5. CONCLUSION

While the development of join algorithms is perhaps one of the best studied problems in database systems research, to date there has been very little published about the intrinsic difficulty of join problems. This may be because joins are too "easy." Any join can be computed in polynomial time — just compute the cross product and iterate through the result applying the join predicate to each tuple in the cross product. While this is true, this coarse-grained analysis belies the experience of researchers who have tried to find truly "good" algorithms for these join problems. As we have shown in this paper, if one "digs down" into the combinatorial complexity of these joins and the resulting optimization problems, one finds a surprisingly rich structure. This structure confirms the experience of our community in developing and implementing join algorithms, that is, that equijoins are "easier" than spatial overlap joins or set-containment joins.

A number of interesting problems remain. One of the most intriguing is the following: many join algorithms in practice work by first mapping the input relations R and S into $R_1 \ldots R_m$ and $S_1 \ldots S_n$, and doing the join by investigating a subset of the joins $R_i \bowtie S_j$, where $1 \leq i \leq m$ and $1 \leq j \leq n$. This is done either to explore parallelism or to make better use of main memory (although with today's memory sizes the second reason is becoming less important.) Here it is natural to ask how hard it is to find the optimal mapping of the tuples of R and S to the R_i and S_j. For the three classes of joins we consider in this paper (equijoins, spatial overlap, set containment), this problem is NP-complete. However, we conjecture that the problem for equijoins has good approximation algorithms.

6. REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *33rd Annual Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[2] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[3] O. Günther. Efficient computation of spatial joins. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 50–59, 1993.

[4] F. Harary. Graph theory. Addison-Wesley, 1969.

[5] S. Helmer and G. Moerkotte. Evaluation of main memory join algorithms for joins with set comparison join predicates. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 386–395, 1997.

[6] T. H. Merrett, Y. Kambayashi, and H. Yasuura. Scheduling of page-fetches in join operations. In *Very Large Data Bases, 7th International Conference, Proceedings*, pages 488–498, 1981.

[7] G. Neyer and P. Widmayer. Singularities make spatial join scheduling hard. In *International Symposium on Algorithms and Computation (ISAAC), 8th International Symposium, Proceedings*, pages 293–302, 1997.

[8] J. A. Orenstein. Spatial query processing in an object-oriented database system. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, pages 326–336, 1986.

[9] C. H. Papadimitriou. Computational complexity. Addison-Wesley, 1994.

[10] C. H. Papadimitriou and K. Steiglitz. Combinatorial optimization: Algorithms and complexity. Prentice Hall, 1982.

[11] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 229–234, 1988.

[12] C. H. Papadimitriou and M. Yannakakis. The travelling salesman problem with distances 1 and 2. In *Mathematics of Operations Research*, pages 1–11, 1993.

[13] J. M. Patel and D. J. DeWitt. Partition based spatial-merge join. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 259–270, 1996.

[14] K. Ramasamy, J. M. Patel, J. F. Naughton, and R. Kaushik. Set containment joins: The good, the bad and the ugly. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 351–362, 2000.

[15] M. Stonebraker. Object relational DBMS: The next great wave. Morgan Kaufmann, 1996.